

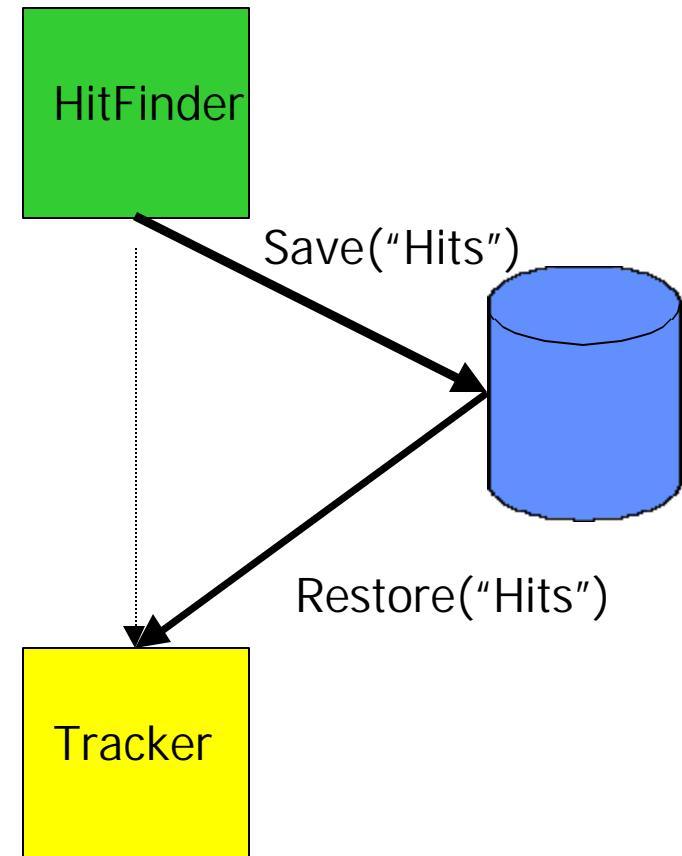
An aside: the Framework and the Event

- â Christian, RD, Simona: prototype of raw event decoding and indexing
- â How about transient physics objects and collections (e.g. Tracks, Pions) ?
- â Large domain, focus on framework specifics:
user interface for physics objects
(how do I get hold of a track collection inside my module?)



GAUDI approach

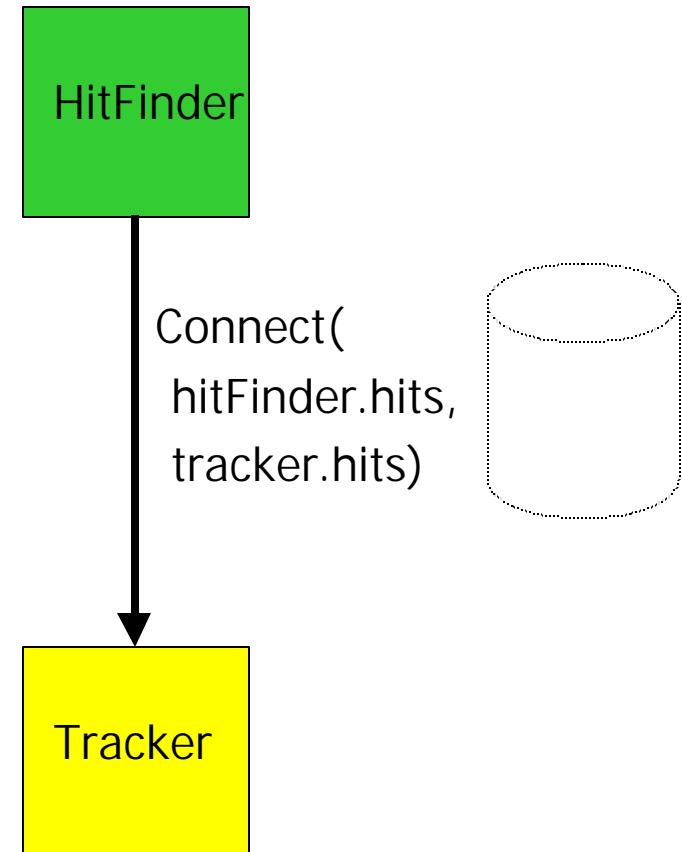
- â the hierarchical DB.
 - No data interface.
 - No static type check
(type laundering, dyn-
cast).
 - Geant(PAW)-like
addressing,
/ATLAS/TILE/WED/...
 - Easy to understand, but
hard to maintain.



Object Network Approach:

â Data flow architecture.

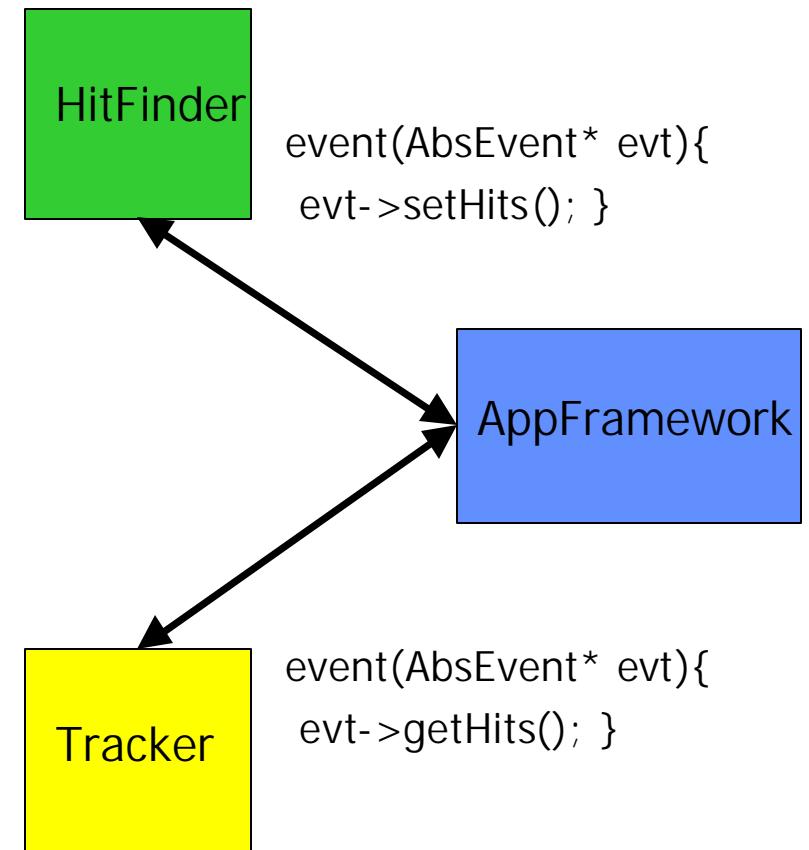
- No (role of) event store.
No global memory management.
- Strict interface.
- Clean but tough.



AC++ Approach

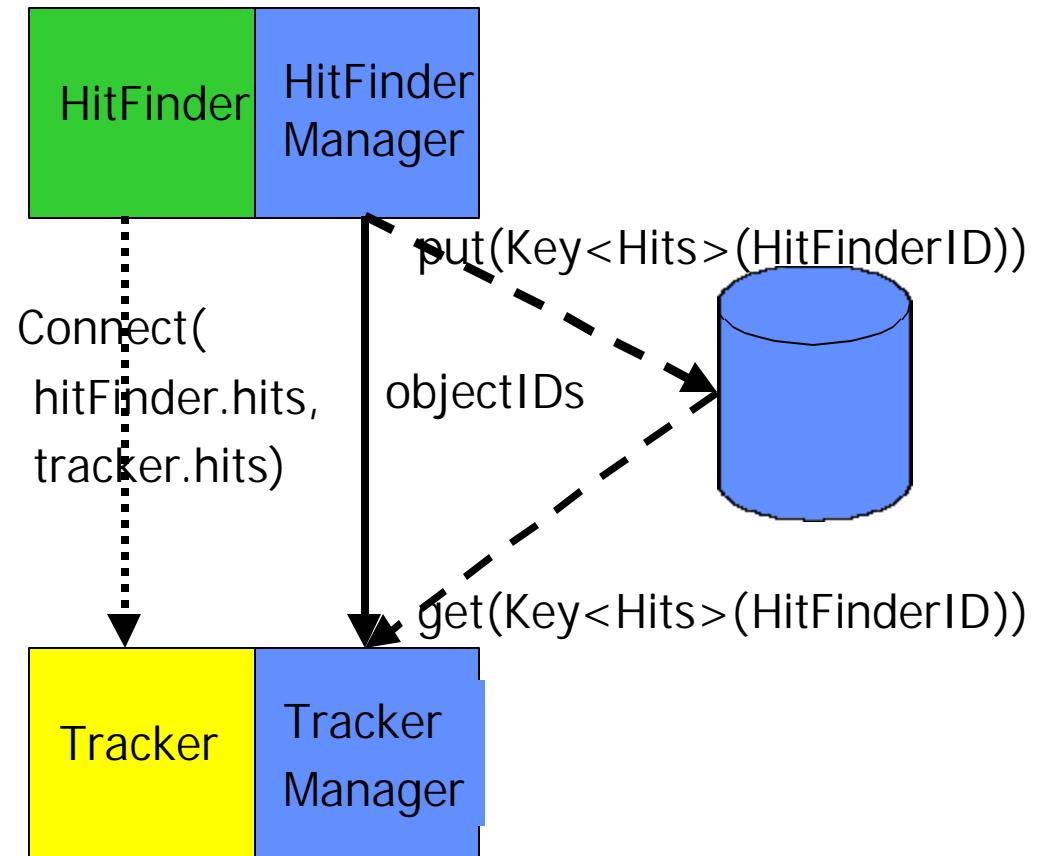
â the c++ approach.

- Abstract event as module argument.
- Hidden interface.
- Often broken interface
`it=obj1()->obj2()->getIt()`



Simulated Data Flow

- â Event: a set of **growable** collections (D0/CDF)
- â Collection authors (may) define specific **selectors**.
- â `get(Key<T> key)` returns a **persistable** collection of object Ids
- â Modules own and **pass around object Ids**, not (pointers to) object!



Simulated Data Flow features

- â Exploit a **module interface dictionary** to generate managers' code (STAF)
- â Allow **direct access to event collections (typed, no dynamic-cast)** for data analysis and quick and dirty development
- â Easy **migration** from direct access **to a clean interface** (replacing the “get” with a module argument in the interface dictionary)
- â **Efficient handling of complex selections** by passing around the **collection of IDs** (even among modules belonging to different jobs)
- â Easy to address multi-language issues



A Toy Implementation

```
class HitFinder__newEvent : public virtual IRunnable {
    .....
    //IRunnable implementation, usually generated from dictionary
    Result run(const IScheduler& s) {
        Result rc;
        Handle<TrackSet> set1;
        Handle<TrackSet> set2;
        Container<ParticleSet> set3;
        IDSelector C0Tsel(c0RCID) //the unique ID of a module
        IDSelector SVXsel(svxC0CID)
        Key<TrackSet> key1(C0Tsel);
        Key<TrackSet> key2(SVXsel);
        Key<ParticleSet> key3("chargedCandidates", hitRCID);
        event->get(key1, set1); //this could be javaGet of f77Get
        event->get(key2, set2);
        rc = _comp->newEvent(set1, set2, set3);
        if (rc == Result::success)
            event->put(key3, set3);
    }
};
```



Where do we stand?

- â We have a web page
<http://iago.lbl.gov/paolo/ATLAS/framework/acti ondesign.html>
- â We have a **prototype** (can get it from the same URL)
 - Core classes running
 - Interface dictionary starting
 - Scripting in progress (IDL to Swig, John M.)
- â We can use the prototype as a test bed for the requirements and use-cases exercises in progress
- â It's time to discuss an Event Data Model!

